Integrated Design of Wind Power Systems: MATLAB - HAWC2 Interface

Barahona Garzon, Braulio; Andersen, Peter Bjørn; Hansen, Anca Daniela; Cutululis, Nicolaos Antonio; Sørensen, Poul Ejnar

Published in:
Proceedings of SIMS 50

Publication date:
2009

INTEGRATED DESIGN OF WIND POWER SYSTEMS: MATLAB - HAWC2 INTERFACE

Braulio Barahona,* Peter B. Andersen, Anca D. Hansen,
Nicolaos A. Cutululis and Poul Sørensen
Risø-DTU National Laboratory
Wind Energy Department
P.O. Box 49, DK-4000 Roskilde, Denmark

ABSTRACT

This paper explores basic concepts for coupling two dedicated software packages in order to simulate the dynamics of wind power systems. The objective is to develop an integrated simulation environment that combines dedicated tools for mechanical and electrical analysis, control and power system integration. This work presents the interfacing of Matlab/Simulink and HAWC2. The latter is an aeroelastic simulation tool, developed at Risø DTU. The technical possibilities of interfacing with Matlab and Simulink are described and discussed. An integrated model of a fixed-speed wind power system is presented. It includes structural, aeroelastic, mechanical and electrical systems of the wind turbine. The electrical components are modeled in Matlab/Simulink; while the structural, aeroelastic and mechanical components are modeled in HAWC2. The dynamic interfacing between Matlab/Simulink and HAWC2 is done using dynamic libraries and TCP/IP. A fixed-speed wind turbine under normal operation is simulated to test the interface.
Keywords: wind power, dynamic simulations, Matlab application programing interfaces, HAWC2, shared libraries, TCP/IP, fixed-speed wind turbine

INTRODUCTION

In the context of higher penetration of wind energy in the power system [1], the wind power plant's ability to provide services such as fault-ride-through and frequency control is becoming increasingly more important. Such operating conditions impose loads on wind turbine mechanical and structural components that may be significant. Furthermore, other services that may be required in the short term, such as storm control will also have an impact on the loads of wind turbines.

Therefore, it is considered relevant to study the dynamical interactions of wind turbines with the grid from an integrated perspective and with the goal of better integration of wind power into the grid. Thus the motivation to couple dedicated tools for aeroelastic design with dedicated tools for control and power system analysis. The first step towards an integrated design simulation platform, is the actual coupling of HAWC2 (Horizontal Axis Wind Turbine Code) with Matlab/Simulink. Figure 1 shows the concept of an integrated design platform, where aeroelastic, structural and mechanical components are simulated in HAWC2 while the electrical components are wind turbine controls are simulated in Matlab/Simulink.

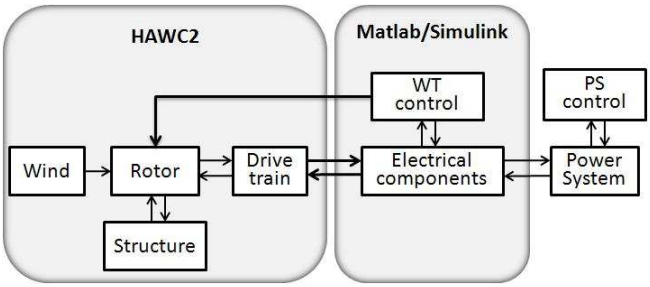

Figure 1: Wind power integrated design platform: HAWC2 - Matlab

HAWC2 is an aeroelastic simulation software, based

*Corresponding author: Phone +45 4677 5092 E-mail: braulio.barahona@risoe.dk

on blade element momentum theory and multi-body dynamics, developed at Risø DTU. It is aimed at simulating wind turbine loads under dynamic loading conditions and control. It is mainly coded in Fortran and licensed as a Windows OS executable. Matlab/Simulink is a well know scientific computing high level language with a graphical environment particularly suited for design and simulation of dynamic system and controls.
In the following Section the available options for interfacing Matlab with other applications and the method implemented to interface it with HAWC2 are described. The section The Dynamic System Under Study describes in a general manner the wind turbine system under study and its dynamics. Following, the results of simulating a fixed-speed wind turbine under normal conditions of turbulent wind are presented. Finally, conclusions regarding the perspective of this interface towards an integrated design platform are presented.

INTERFACING APPLICATIONS

There are several possibilities to interface a given application with Matlab and Simulink. This section explores some of the concepts and methods commonly used to interface with Matlab/Simulink. Fundamentally, making available for Matlab a program or function written in a different language is done with the objective of taking advantage of the given application for use in Matlab programs. In some cases, saving computation time can be the driver for accessing shared libraries. Also the matrix manipulation, visualization and control tools in Matlab are attractive to other applications.
For example, the interfacing of GAMS (optimization software) and Matlab as described in [2], is done in order to combine the GAMS ability for large nonlinear optimization with the Matlab visualization tools. This is achieved by having a *gams mex interface*, which basically comes down to a shared library compiled as Matlab executable (MEX-file) that is available from the Matlab command line as any other built-in function.
Another example is how PSCAD/EMT (electromagnetic transients simulation program) and Matlab are coupled[3]. This interface takes advantage of interprocess communication capabilities in both software using a Fortran subroutine (for PSCAD/EMT) and the "Matlab engine" for Matlab. The Fortran subroutine starts a Matlab engine and initiates the communication process, making in this way the relevant Matlab mathematical functions and control system blocks (written in a Matlab m-file) available to PSCAD. This interface also allows the Matlab engine to run on a different computer on the same network, thereby allowing *parallelism* of the computation time.
Another way of interfacing with Matlab is the so-called *parallelization* of Matlab. In the sense that some of the tools used in order to run various Matlab engines (slaves) in a cluster called by another Matlab engine (master), can be attractive to interface Matlab with other applications. There has been more than a few initiatives to develop a parallel Matlab [4]. Nowadays, Matlab provides a Distributed Computing Server interface that supports a variety of platforms and operating systems. Also other platforms can be integrated using other generic application programing interfaces (APIs) [5].
The power system simulation tool PowerFactory, also provides possibilities of interfacing with Matlab/Simulink. Mainly, with the objective of taking advantage of the control design capabilities in Simulink [6].

Options for interfacing with Matlab/Simulink

Shared libraries

The concept of *shared libraries* can be ambiguous to some extent. In this document a shared library is considered a collection of functions that can be dynamically loaded by Matlab at run time [7]. The objective of shared libraries is that various programs can access a given set (i.e. library) of functions, subroutines, classes, etc., coded in another language (C, C++, Fortran, Delphi). It can be said that a shared library consists of a header file and the library itself (a `*.dll` file in Windows OS). The header file provides the *signatures* (or *prototypes*) of the functions, in other words the declaration of the function's name, type and number of arguments. Where as the library file provides the actual definitions of the functions.
In order to give Matlab access to an external function in a shared library the user needs to:

- load the shared library, and

- call the desired function.

This simple process involves knowing the full path of the library and the header file. It is then necessary to know the signature of the shared library functions (SL functions) and the type of arguments to be passed to it. Some argument types in Matlab are, in practical terms, the same as the corresponding type in the language the library is programmed. However, if the types are different they can be automatically or manually converted using Matlab functions as described in [7].

MEX-files

Similarly, MEX-files are functions originally written in another programming language (i.e. source MEX-files in C or Fortran) and compiled to be *Matlab executable files* (namely, `*.mexw32` files in Windows OS). Once these binary files are loaded, they work like Matlab built-in functions. These definitions are further explained in [7]. The structure of a source MEX-file is basically:

- comments,
- headers,
- computational routine and
- gateway routine.

A standard gateway routine, named `mexFunction`, is available. This is the actual interface to Matlab. Within `mexFunction` there is a section of declaration of variables, then the inputs and outputs are assigned to the corresponding variables. Finally the computational routine is called, again with arguments corresponding to the input and output variables requested from Matlab. A MEX-file can actually contain a set of functions, therefore it is similar to a shared library, with the difference that the MEX-file functions can be called like native Matlab functions.

S-functions

A system-function (S-function) is a Simulink block written in computer language (i.e. Matlab, C, C++, Fortran) [8]. Basically, it can be implemented as:

- a Level-1 or Level-2 M-file S-function (written in Matlab) or
- as a C MEX S-function (hand written in practically any modern language, via the S-Function Builder or via the Legacy Tool Code).

TCP/IP communication

The Transmission Control Protocol/Internet Protocol is a mechanism for transferring data between applications over multiple networks [9]. The origins of this protocol lie in the predecessor of the Internet and it is the under laying working architecture of the Internet as it is known today. Some of the features of TCP/IP are:

- quick and simple to configure,
- one end is the client and the other server, but once the link has been establish both can send and receive data,
- non-transactional, not persistent (i.e. it uses memory buffer),
- no built in security, and
- no standard way of signaling end and start of message.

It is possible to make applications talk to Matlab using TCP/IP. Such feature is included in the Instrument Control Toolbox and it is oriented to communicating with instruments that support this protocol [10]. It has been used to communicate with remote applications as described in [11].

Matlab engine

This feature allows external applications to call Matlab software and running it in the background as a separate process. In Windows OS it uses a Component Object Model (COM) interface [7], which enables interprocess communication between applications. The engine library has routines available that allow the external application to control Matlab computation engine. These functions basically come down to start up/shut down and send/receive data.

Matlab - HAWC2 coupling

The task of dynamically interfacing Matlab and HAWC2 for solving complex dynamic systems,

such as a wind turbine, can be approached in various ways. If the objective is to be able to access HAWC2 libraries directly from Matlab or to create shared libraries for HAWC2 directly with Matlab, then some of the options described in the previous Section are applicable. However, the overall objective of this work is to create an interface that allows HAWC2 and Matlab to interact dynamically in order to simulate complex wind power systems, while bearing the possibility of connecting a third application (e.g. DIgSILENT) with Matlab and HAWC2. The solution developed is based on keeping HAWC2 as a stand-alone application while letting Matlab *manage* some HAWC2 processes and have dynamic access to data and input variables every time step. Thereby, an interface as described by figure 2 using dynamic libraries and TCP/IP sockets was implemented and tested. With this interface HAWC2 and Matlab run as separate processes on the same computer, Matlab acts as a client and HAWC2 as a server using local TCP/IP sockets (i.e. local IP addresses and port numbers). Once the connection is established both ends can send and receive data. HAWC2 is set to remaining *listening* for the client to send inputs and requests.

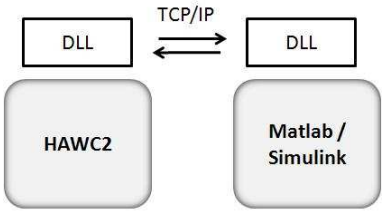

Figure 2: HAWC2 - Matlab interfacing using shared libraries and TCP/IP sockets

THE DYNAMIC SYSTEM UNDER STUDY

A wind turbine is a complex dynamic system subject to stochastic loads driven mainly by the wind. From a general functional perspective, it can be considered to be composed of structure, rotor, drive train, electrical components and control components (figure 1). These systems interact dynamically with each other and with the power system.

For this work an active stall fixed-speed wind turbine was implement. This wind turbine concept is often described in the literature, the main components are: rotor, drive train, generator and pitch control. Typically, the generator is a squirrel cage induction generator (SCIG) directly connected to the grid. Therefore, the speed of the wind turbine rotor is practically fixed to the electrical frequency of the grid [12]. The power is regulated by varying the angle of the blades with the pitch control, for wind speeds below rated this is done to maximize the efficiency. Above rated wind speed the blades are pitched to a stall position (i.e. active stall) to limit the power of the rotor.

In this study the wind turbine is considered to operate under normal conditions and the grid is considered infinitely stiff, thus the dynamics considered in this work are:

- Turbulent wind (based on the Mann model, compliant with IEC61400-1 ed. 3 [13])
- Aerodynamics (state-of-the-art aerodynamics considering dynamic inflow, dynamic stall, skew inflow, wind shear and large deflections [13])
- Mechanical components (rotor and drive train)
- Structural components (tower)
- Electrical generator (reduced order model of SCIG based on [14] and [15])
- Control (a pitch servo modeled as a first order filter)

Turbulent wind, aerodynamics, structural and mechanical components are simulated in HAWC2. The generator model and the control are implemented in Matlab. The generator model is a reduced order model formulated in state-space and solved with standard differential equation solvers available in Matlab. The state-space equations are referred to the *dq0* frame, in order to avoid time-varying inductances [16]. The flux linkages of the stator and rotor are selected as state variables, however, the transient of stator fluxes are neglected as described in [14].

The pitch control implemented is described in figure 3, it consists of a moving average of the wind speed with a 60 seconds window, a sample hold of the moving average every 10 seconds, a look up table for the optimal pitch as a function of the wind speed and a pitch servo that is modeled as a first order filter. This simple control strategy works for this

case study where the mean wind speed is kept constant at the nominal value of the machine (12 m/s) and standard operating conditions are assumed. The objective is to test the qualitative performance of the HAWC2 - Matlab interface.

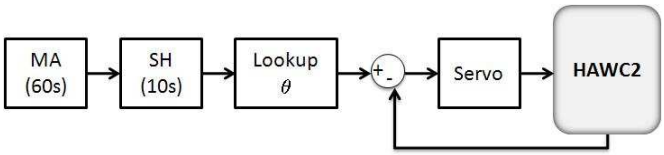

Figure 3: Pitch control for active stall under nominal operating conditions

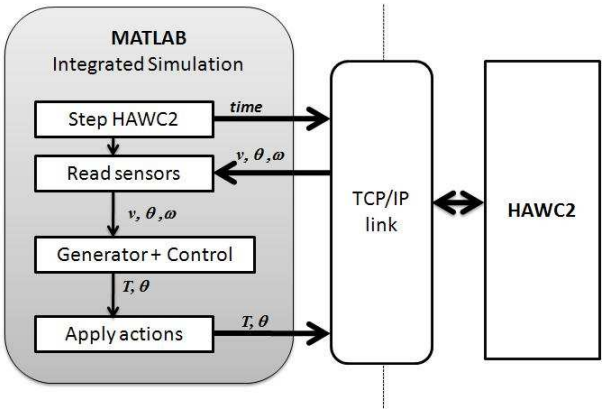

Figure 4: Integrated simulation simplified flow diagram

Integrated simulation

An active stall, fixed-speed, 2 MW wind turbine is simulated with the dynamics described to test the performance of the interface. The simplified flow diagram in figure 4 describes the integrated simulation computational flow.

Previous to the flow depicted in figure 4, time-data management variables and generator model are initialized in Matlab. Then the shared library developed to interface with HAWC2 is loaded and functions of this library are used to startup HAWC2. At startup, HAWC2 uses specified input files and sets a local TCP/IP socket listening to Matlab. Matlab then asks HAWC2 to step to a given *initialization* time and wait.

Once the systems in both applications are initialized, the simulation continues with Matlab reading information of the relevant states (shaft speed ω) and variables (wind speed v, blade angle θ) from HAWC2, then solving the generator model and estimating a control input. The output of the generator model (torque T) and the control input (blade angle θ) are sent back to HAWC2 along with the indication of iterating to find the solution of the next time step. HAWC2 solves its system, sends states and variables back and waits again for Matlab. The process is then repeated, keeping the same time step size in both simulation tools, until the end of the simulation.

SIMULATION RESULTS

This section presents a simulation of the active stall fixed-speed wind turbine, previously described, under normal operating conditions. Figure 5 shows the wind speed perpendicular to the rotor at hub height (80 meters), the pitch angle of the blades and the power produced by the wind turbine. Figures 6 show the tower top and bottom bending moments and figure 7 the flapwise (M_x) and edgewise (M_y) blade root moments.

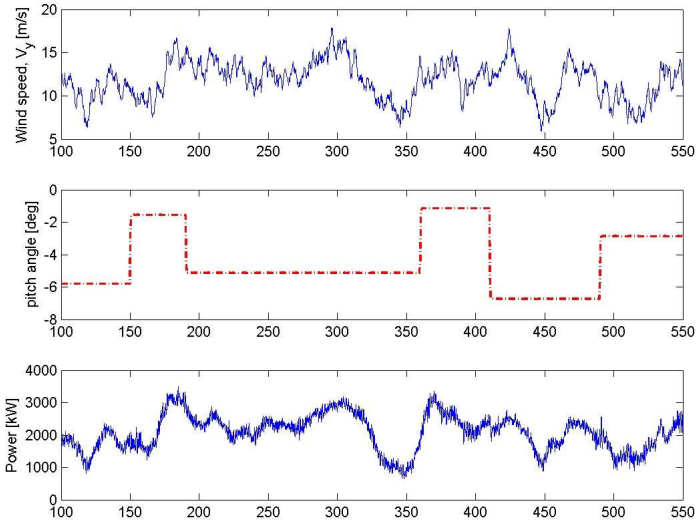

Figure 5: Wind speed, pitch angle and power

CONCLUSION

An interface for Matlab/Simulink and HAWC2 based on shared libraries and TCP/IP sockets has

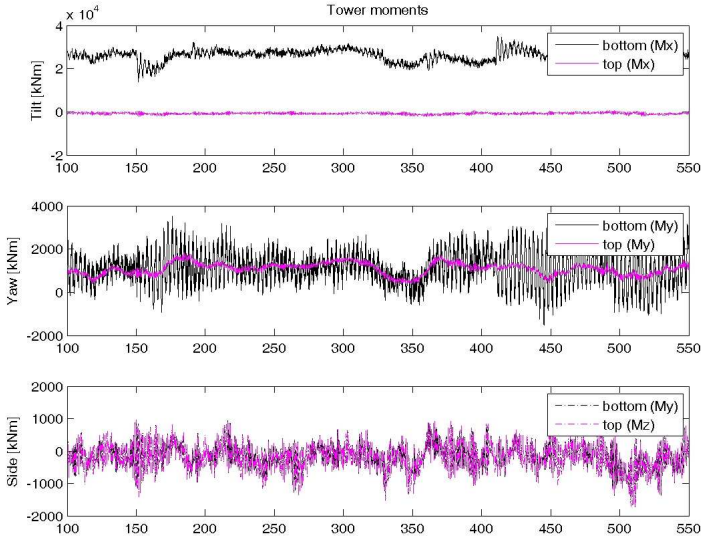

Figure 6: Tower bending moments

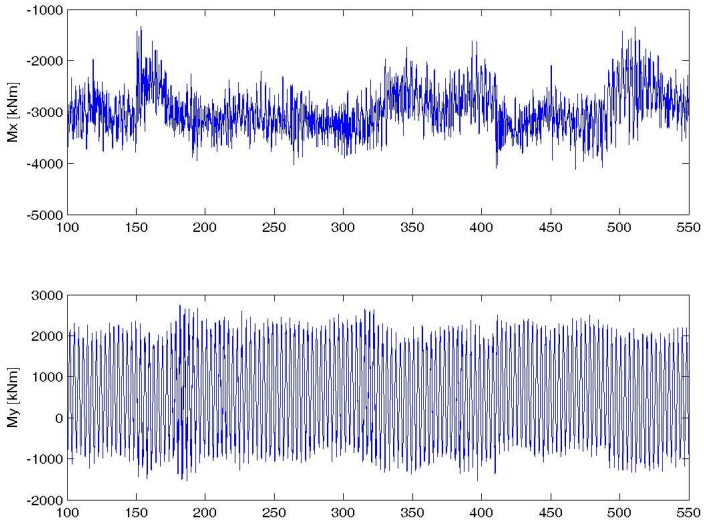

Figure 7: Blade root moments

been introduced. This interface lets Matlab interact with HAWC2, making possible to perform dynamic simulations with models written in Matlab/Simulink. The interface was tested qualitatively with a fixed-speed active stall wind turbine under normal operation conditions. Generator and pitch control were developed in Matlab while turbulent wind, aerodynamics, structural and mechanical components were implemented in HAWC2.

The interface developed takes advantages of the capabilities and tools built-in in Matlab for interfacing with external applications. It combines such tools in a particular manner specific to this problem. The next step is to increase the complexity of the system and the control to simulate wind power systems under conditions that are relevant for grid integration issues. In this stage, it is considered possible to extend the interface described here to include a power system simulation tool. However, it will very much depend on the built-in interfacing capabilities of the software, particularly when it comes to commercial software.

ACKNOWLEDGMENT

This paper describes the ongoing results of a EFP project titled "Integrated Design of Wind Power Systems". The Danish Energy Agency is acknowledge for funding this work in contract number EFP07-II.

The work is carried out by the Wind Energy Department at Risø-DTU National Laboratory in cooperation with Aalborg University.

REFERENCES

[1] "IEA WIND ENERGY Annual Report 2007," tech. rep., International Energy Agency, July 2008.

[2] M. C. Ferris, "Matlab and GAMS: Interfacing Optimization and Visualization Software," *University of Wisconsin, Computer Science Department*.

[3] A. Gole and A. Daneshpooy, "Towards Open: Systems A PSCAD/EMTDC to MATLAB Interface," in *ISPT '97*, June 1997.

[4] R. Choy and A. Edelman, "Parallel MATLAB: Doing it Right," *Proceedings of the IEEE*, vol. 93, pp. 331–341, February 2005.

[5] The MathWorks Inc., *MATLAB Distributed Computing Server 4.0*.

[6] DIgSILENT GmbH, *DIgSILENT Technical Documentation - PowerFactory DSL Models*, August 2008.

[7] The MathWorks, Inc., *MATLAB External Interfaces*, online edition ed., October 2008.

[8] The MathWorks Inc., *Writing S-functions*, on line ed., October 2008.

[9] A. S. Tanenbaum, *Computer Networks*. Pearson Education International, 4 ed., 2003.

[10] The MathWorks, Inc., *Instrument Control Toolbox*, July 2005.

[11] E. J. Mayhew, "Building a Distributed Aircraft Tracking Simulation Using TCP/IP," *MATLAB Digest*, July 2005.

[12] I. Munteanu, A. I. Bratcu, N. A. Cutululis, and E. Ceangă, *Optimal Control of Wind Energy Systems*. Springer, 2008.

[13] T. J. Larsen, "How 2 HAWC2, the user's manual," tech. rep., Risø DTU, January 2009.

[14] F. Iov, A. D. Hansen, P. Sørensen, and F. Blaabjerg, "Wind Turbine Blockset in Matlab/Simulink. General Overview and Description of the Model," tech. rep., Aalborg University, 2004.

[15] N. A. Cutululis, T. J. Larsen, P. Sørensen, F. Iov, and A. D. Hansen, "Electrical Components Library for HAWC2," tech. rep., Risø DTU, 2007.

[16] P. C. Krause, O. Wasynczuk, and S. D. Sudhoff, *Analysis of Electric Machinery and Drive Systems*. IEEE Press Series on Power Engineering, WileyBlackwell, 2002.